RDBM versus OODBM in Support of Integrated Data Bases for Computer Aided Building Design

Jin-Yeu Tsou, James A. Turner, Harold J. Borkin[1]

Introduction

Computer technology has changed the way architects and engineers design, analyze, plan, estimate and document buildings. In the last few years powerful and affordable software and supporting hardware have replaced traditional tools, making modeling, engineering calculations and drawing quicker and easier. Software designers and researchers are constantly forecasting an even brighter future, with cheaper and better computer applications available to the practicing architect. This trend will continue; but the promises made in the 60's and 70's, of a thinking, computer design assistant, or of a completely integrated collection of architectural design, analysis and drawing tools, may take longer than first forecast.

A critical issue which must be investigated for the "next generation" of computer aided building design applications is integrated building data bases. These data bases must be dynamic and extendable to support new types of information; must be shareable by architectural designers and various types of engineers; must be able to support collaborated design activities; must be able to effectively represent complex relationships among design objects; and must provide enough power for rich data modeling. The benefits of an integrated system have been recognized and reported as an important issue for many years [Eastman 75, Borkin 78]. Conventional data base management systems (e.g., network, hierarchical, or relational) may not be adequate to support these requirements, but advancements in data base technology make it possible to now address the problem.

This paper will compare two data base management technologies – the relational data base and the object-oriented data base – as candidates to support an integrated architectural CAD environment. In order to investigate these two approaches, a simple architectural design scenario and conceptual building model are presented. The relational and object-based approach will be implemented from the scenario and model. The paper will compare the inherent qualities of the two systems, and will comment on their possible adaptation to the traditional and emerging, computer-assisted, practice of architecture.

Overview of RDBM and OODBM Systems

Conventional relational data base management (RDBM) systems have served the needs of a variety types of business applications which have pre-defined and static data requirements, such as accounting and inventory control. These systems have been suggested as a possible solutions to future architectural CAD systems [Borkin 78, McIntosh 84].

In recent years, object-oriented data base management (OODBM) systems have been developed to meet the goals of data modeling, schema evolution, performance, cooperative design, and version management [Joseph 91]. OODBM systems, suggested by the research of object-oriented information management [Ahad 92, Bertino 91, Cattell 91, Kim

---

[1]Architectural Research Laboratory, College of Architecture and Urban Planning, The University of Michigan

91, Ketabchi 90, Kim 90], may provide a sound data base foundation for the next generation of CAD applications.

Conventional RDBM System

Because of the acceptance and standardization of relational data base systems, many computer-aided architectural applications use either an RDBM system for their information storage, or provide an interface to a commercial RDBM system. An RDBM system is based on the mathematical theory of relations; to the casual user, the information is organized in tables. The columns of these tables represent the domains of possible values, and the rows represent a record or a set of values for each column. Relational data base systems offer data base definition languages and data base manipulation languages for table manipulation. Most RDBM systems conform to Codd's [Codd 79] definition of a fully relational system:

a. A relation consists of a set of tuples with each tuple having the same set of attributes.
b. Each attribute is chosen from a domain of possible values. If the domains are simple the relation can be represented as a table.
c. There are no duplication of rows.
d. Row and column (attribute) order are insignificant.
e. All table entries are atomic.
f. A set of operators exist which operate on one or more relations and produce a relation as result. Operators include union, intersection, difference, symmetric difference, selection, restriction, projection and join.
g. To maintain consistency values must pass a domain candidacy test before being put into or being taken from a relation. Domain candidacy tests are typically a simple membership conditions which can prevent values which are out of range from being added to a table such as a number which is too large or a misspelled color name.

Concepts of the Object-Oriented Approach

The object-oriented approach is becoming increasingly popular in the design and development of computer software applications. This is because the object metaphor provides a natural way to map real world objects and their relationships directly to computer representations. The fundamental concepts of the object-oriented approach are objects, attributes, methods, classes, and inheritance. These concepts first appeared in object-oriented programming languages, and many of the ideas date back to the development of Simula-67 [Dahl 66].

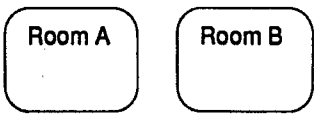
a. Objects are entities used to represent abstract or concrete real-world things in an application domain.
b. The internal state of an object is a set of attribute values, and its behavior is a set of methods (procedural knowledge) which operate on the state of the object.
c. A class is a template for creating objects which share a common set of attributes and methods.
d. Classes exist in a hierarchy of classes and inherit properties from their parents.

Computer Aided Building Design Scenario

A simple design session scenario is illustrated to highlight the needs and functions of architectural information. Given at each stage is: a design action; the kind of information inquired from the data base; the procedures to be performed to evaluate alternatives; and the

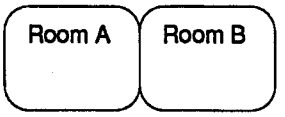
types of semantic structures needed to represent the entities and relationships. The following is a possible scenario of actions involving the design of two connecting rooms:

Design Action 1 Room A is added. Room B is added.



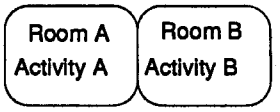
Data-Base Activity: Two new room instances with default attributes are added.

Design Action 2 Room A is adjacent to Room B



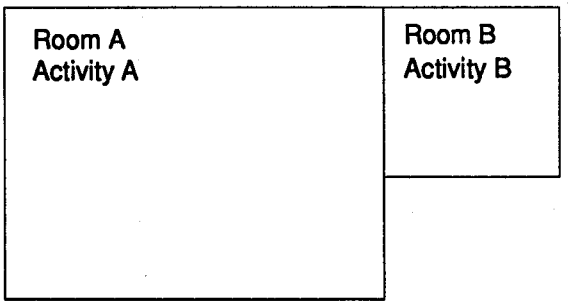
Data-Base Activity: An adjacency relationship is established between the two rooms.

Design Action 3 Room A given activity. Room B given activity



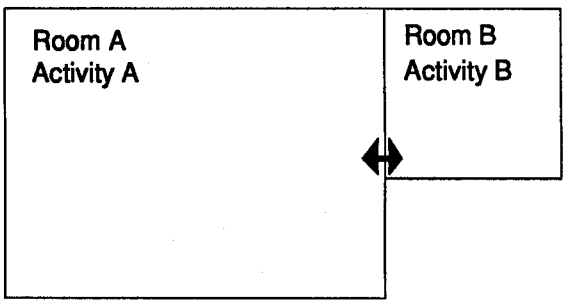
Data-Base Activity: Activity property added to rooms A and B. At this point a judgement is made as to the suitability of having activity A adjacent to activity B. A Building Program data base (see **Building Program Data Base**) is consulted for possible violations (see **Violation Resolution**).

Design Action 4 Room A given polygonal shape. Room B given polygonal shape.



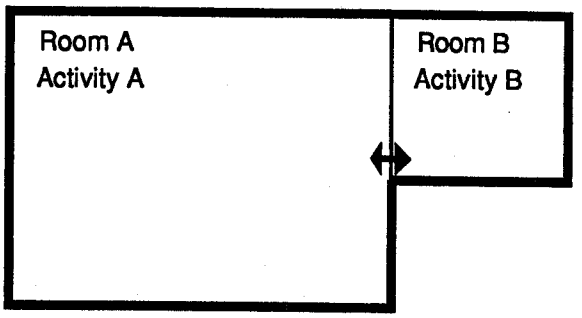
Data-Base Activity: A geometry attribute (polygons) is added to both rooms A and B. Wall instances coinciding with edges of the polygon and with default values are added to rooms A and B. At this time any geometric requirements from the Building Program for activities A and B can be checked (such as minimum dimensions, minimum area, shape, etc.).

Design Action 5 A door is added between room A and room B



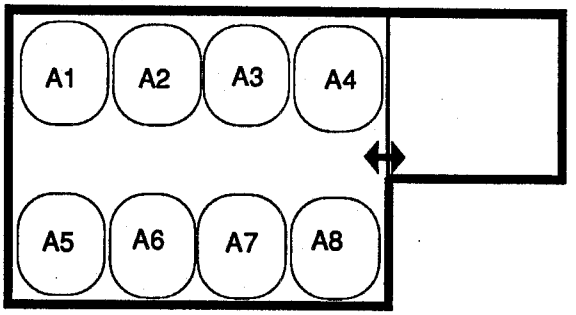
Data-Base Activity: One new door instance with default attributes is added; A relationship is established between the door instance and one of the walls.

Design Action 6 Walls assigned assembly type.



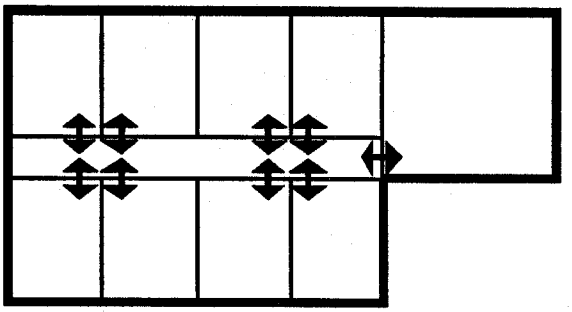
Data-Base Activity: A material assembly attribute is added to each wall instance for rooms A and B. At this time the activities can be re-evaluated; that is, it may now be acceptable to have the two activities adjacent given the wall type of the dividing wall. (Or, the necessary wall finish dictated by one of the activities may be impossible to attach to the wall assembly.)

Design Action 7 Rooms added to Room A.



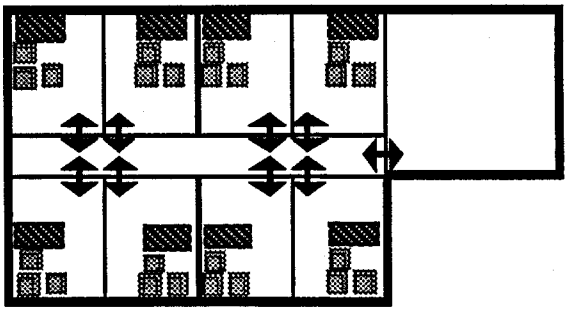
Data Base Activity: A relationship between room A and rooms A1 - A8 is made; that is, rooms A1 - A8 are "contained in" room A. The Building Program is checked to determine if it is acceptable to add rooms to room A.

Design Action 8 Added rooms given shape (walls and wall assemblies) and connections.



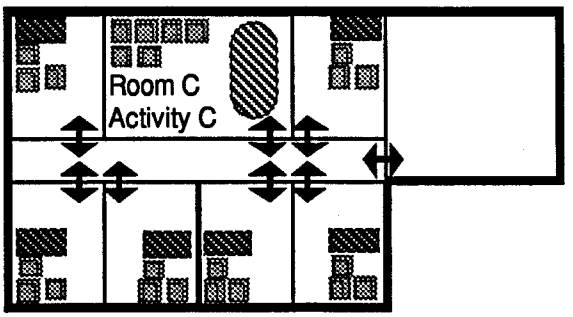
Data Base Activity: Geometry attributes (room polygons) and wall instances are added to rooms A1 - A8. One new door instance with default attributes is added for each room; A relationship is established between each door instance and one of the walls. An adjacency relationship is established between rooms which share walls.

Design Action 9 Furniture added to rooms in room A.



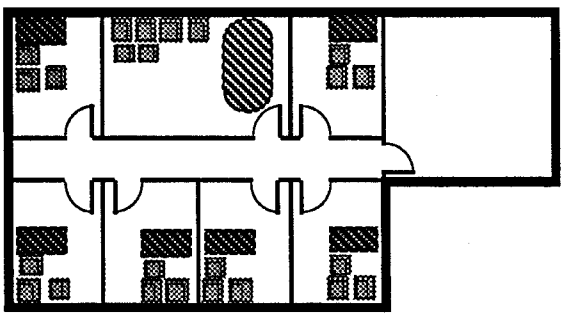
Data Base Activity: An instance is added for each piece of furniture. Only generic types are added - not specific products. The furniture becomes members of the room but have no specific locations or orientation. A check can be made to determine the suitability of the furniture types and number to the activity type of room A.

Design Action 10 Wall removed between room A2 and room A3. Combined room is named room C and assigned activity C. Furniture removed from room C and new furniture added. One door is removed.



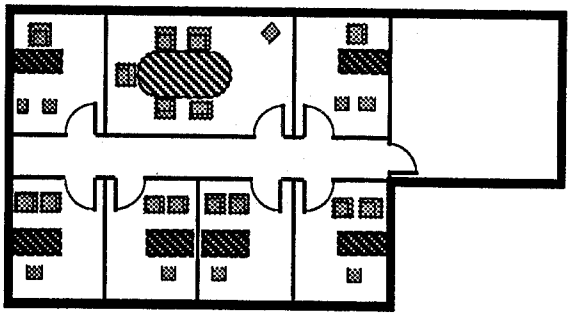
Data Base Activity: Room A3 is deleted; Room name A2 is changed to C; Activity in room is changed to C; Furniture in rooms A2 and A3 is deleted; The door in room A2 is deleted; The wall between room A2 and the corridor (what's left of room A) is deleted; The remaining wall is lengthened; New furniture is added to room C. These changes invoke many Building Program checks.

Design Action 11 Doors given types and locations.



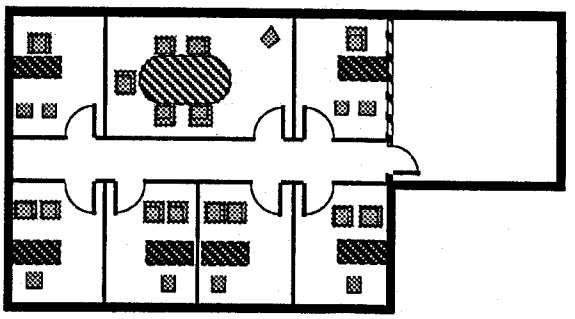
Data Base Activity: Each door instance is given location and opening type attribute values. A check must be made to determine the compatibility of the selected wall material assembly and the door attributes.

Design Action 12 Furniture instances given location and orientation.



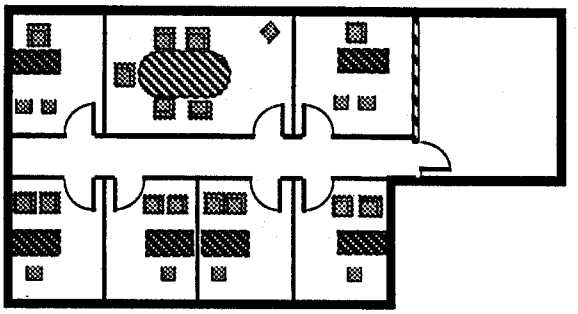
Data Base Activity: Each furniture instance is given location and orientation. An interference check is made between furniture instances and door instances.

Design Action 13 Material assembly of dividing (between rooms A and B) wall is changed



Data-Base Activity: The new wall material assembly needs to be re-evaluated based on the Building Program and the activities of rooms A and B. The door attributes must be checked against the new wall material assembly; that is, it may not be possible to fit the door instance into the new material assembly.

Design Action 14 Dividing wall repositioned



Base Activity: The wall location attribute is changed. Because the room polygon is interference changed, any area or minimum dimension requirements for activity A must be checked. An Data-check must be made between the furniture instances and the repositioned wall.

Conceptual Building Model

Information analysis and classification play important roles in the design of data base management systems. The AEC Building System Model [Turner, 90] is used to conceptually represent the semantic structure of building domain information. Applying such a formal analysis method to the problem domain is essential to the success of

information system design; a step that is lacking in the development of most systems [Abudayyeh, 91]. The design of information systems at the conceptual level can help system designers concentrate on a domain's semantic structure, instead of considering data structures or representation schemes which can be limited by the paradigm of an implementation platform.

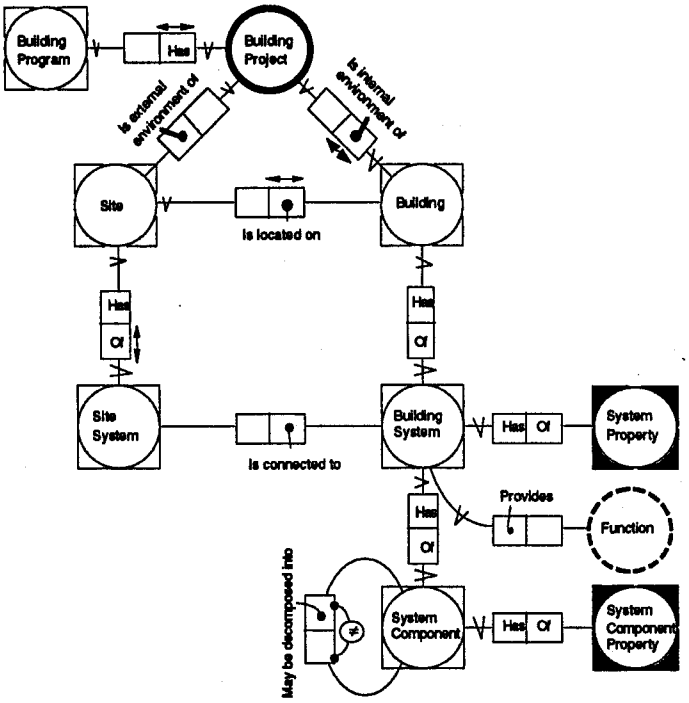


Figure 4 General Building System Model

The AEC Building System Model is a byproduct of the PDES/STEP standardization effort, and is the only national attempt to study an integrated information framework for architecture, engineering, and construction (AEC) information [Peters 91]. The model present a high-level conceptual schema of an AEC product model. The product in this case is a building project which consists of a finished, occupied building and a site.

The AEC Building System Model is based on a general building systems model which assumes that all building components are members of one or more building systems. The model is graphically presented using the Nijessen Information Analysis Method (NIAM) [van Griethuysen 92, Turner 90]. Each system follows a general framework and contains components and properties. For example, one property of an enclosure system might be the cost of the system. A system component may be an exterior enclosure assembly, which in turn could contain other components such as windows and doors. The general building system model is illustrated in Figure 4.

The design scenario manipulates only a few building components – spaces, walls, furniture, and doors. Several conceptual data models, based on the AEC Building System Model, are developed in this section to represent the semantic framework of these components. The semantic framework of objects includes: properties, an identifier, relationships and roles with other objects, hierarchical relationships, and role constraints and cardinalities. This information represented by these conceptual models will be used in both the design of the RDBM and OODBM systems.

Enclosure System and Spatial System Model

Because of the close relationship between occupied spaces and their associated enclosure systems in a building, the two building systems are modeled together. Figure 4.1 illustrates the two systems. For example, the following design information can be derived from the model: there are three kinds of occupied interior spaces - service, assigned, and circulation; an enclosure system component can either be a wall, floor, ceiling or roof.

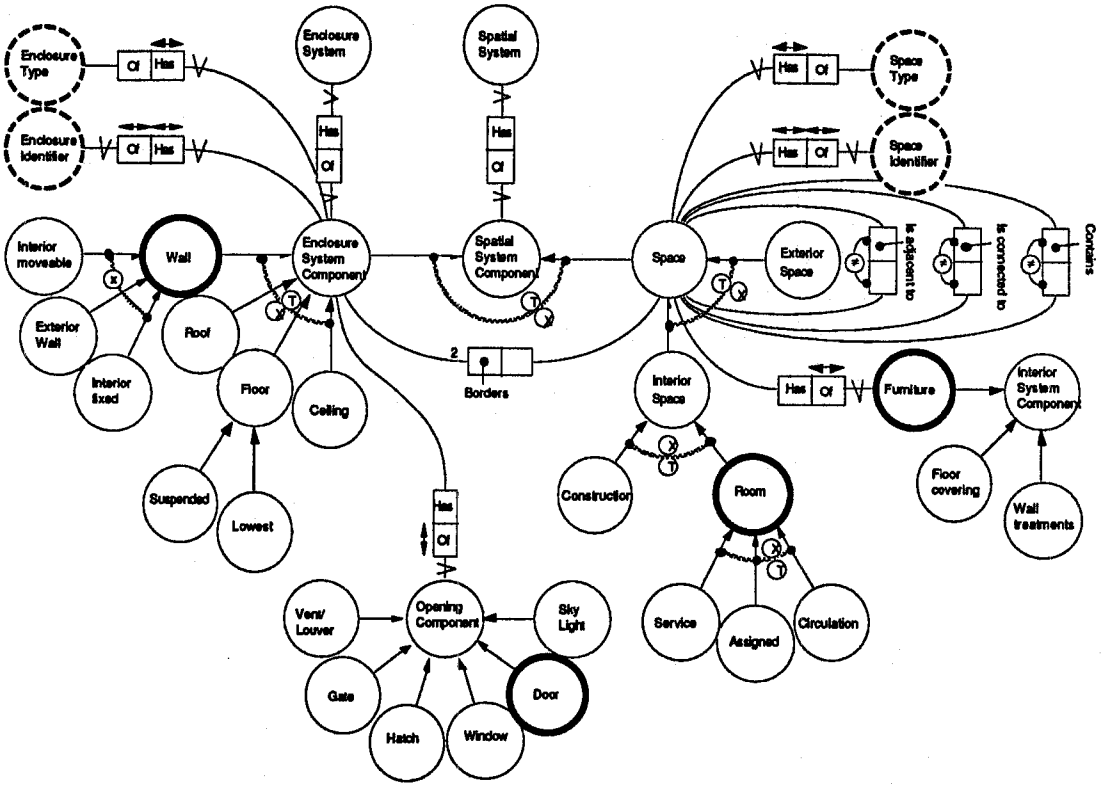


Figure 4.1 Model of Spatial and Enclosure Systems

Room Model

Figure 4.2 illustrates the properties of a room.

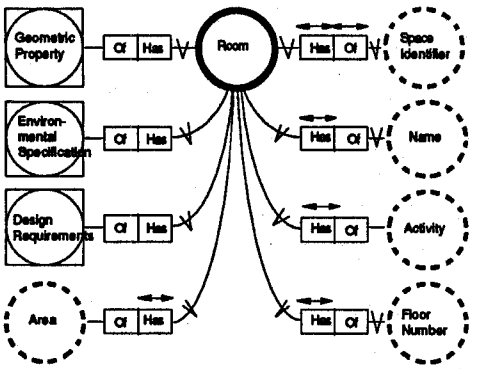


Figure 4.2 The Properties of a Room Instance

Wall Model

Figure 4.3 illustrates the taxonomy of a wall and its properties.

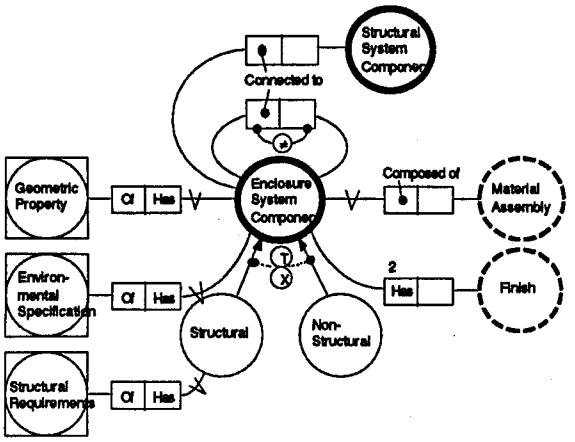


Figure 4.3.1 The Properties of an Enclosure Component

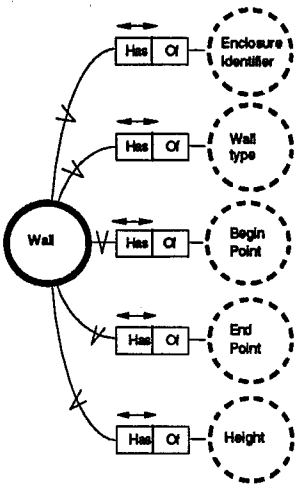


Figure 4.3.2 The Properties of a Wall Instance

Door Model

Figure 4.4 displays a possible taxonomy of an opening and the properties of a door.

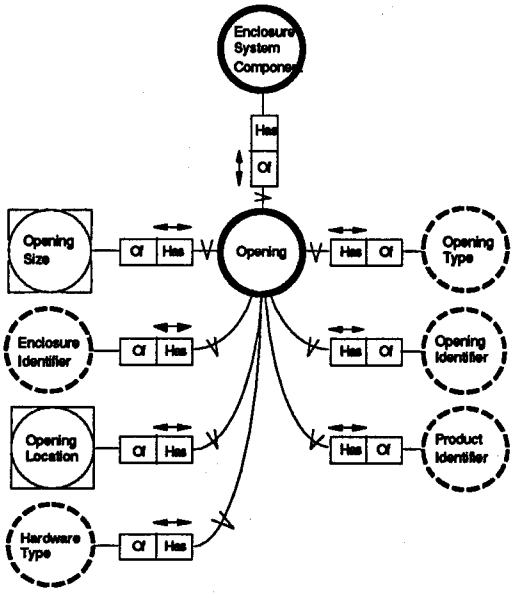


Figure 4.4.1 The Properties of an Opening Instance

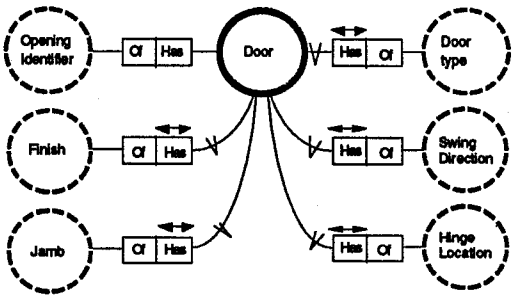


Figure 4.4.2 The Properties of a Door Instance

Furniture Model

Figure 4.5 shows the properties of furniture instances.

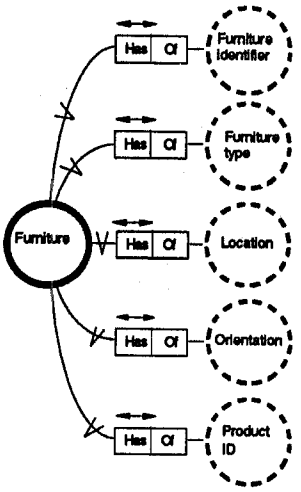


Figure 4.5 The Properties of a Furniture Instance

Violation Resolution

Quite often additions, deletions and modifications to a data base result in violations caused either by data inconsistencies or by incompatibilities with a building program data base. To stop until each violation is resolved would hinder the flow of the design process. While in the end each violation will have to have been addressed and resolved, during the process the violations need only to be remembered.

As a possible solution to remembering violations the following extension to the model is proposed:

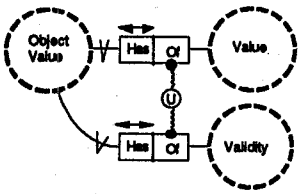


Figure 4.6 Object Value Validity

Violation values can either be categorical (success, failure) or interval (continuous value). The validity value is proposed to be a fraction between 0 and 1. A value of 1 being highest confidence in the value (passes all checks) and a value of 0 being the lowest confidence value (must be resolved). Values between 0 and 1 would represent a scale of validity. For instance, if the building program data base allowed activity A to be adjacent to activity B, a validity of 1 would be assigned. If the data base stated that under no circumstances should activity A be adjacent to activity B, a validity of 0 would be assigned. The Building Program data base could also suggest a validity value for the adjacency of activities A and B, say .3.

A CAD program could evaluate the design solution periodically, either automatically or under user control, by checking validity values for all object values. Of course, not all object values need a validity value.

Building Program Data Base

Most of the data base activities from the scenario involve comparing the results of a design action with an rule or declaration in the "building program" data base. The following two models represent part of a building program data base which is used for design actions 3 and 4:

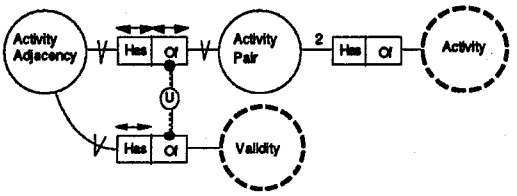


Figure 4.7.1 Building Program Model Activity/Adjacency

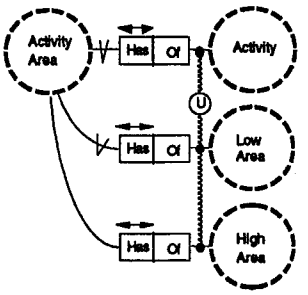


Figure 4.7.2 Building Program Model Activity/Area

The Activity/Adjacency data base is a list of pairs of activities and a code signifying the validity of the pair of activities being next to each other. The Activity/Area data base is a list of activities and the low high values the area of the room polygon with the activity should be between.

These two models are only a small part of what a complete building program data base must be in an integrated computer aided building design system. These two models are mapped into tables in the implementations, but many building program criteria would have to be in other forms, such as single values and rules.

Data Base Activity Algorithms

The data base activity associated with design action 3 is the first to query the building program data base. When an activity is assigned to a room, the activities of all adjacent rooms are tested against the building program Activity/Adjacency table.

When a polygonal description is assigned to a room, as in design action 4, the room area is computed and stored, the area is compared to the acceptable range of areas for the room activity, and a wall is added for each polygon edge:

Relational Data Base Design

The RDBM approach begins by defining property tables and relational tables for each domain entity. The NIAM conceptual model is directly mapped to relational entity property and entity relational tables.

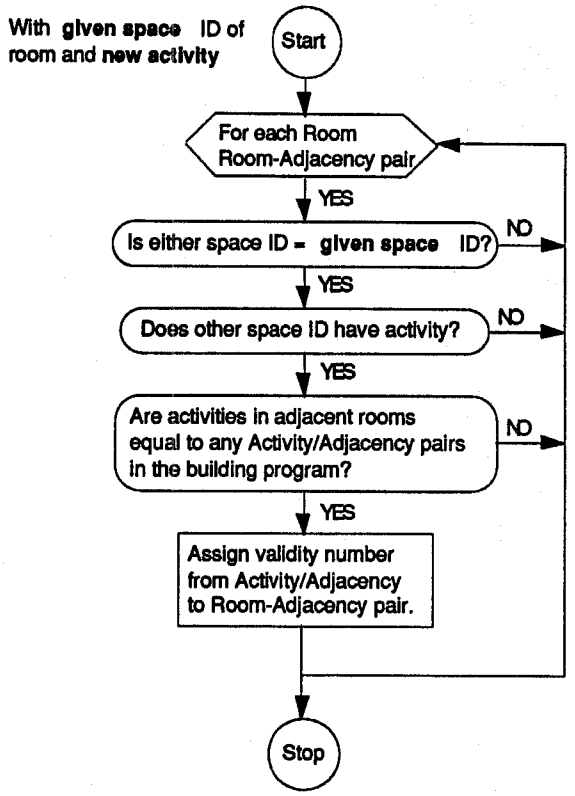


Figure 4.8.1 Data Base Activity 3

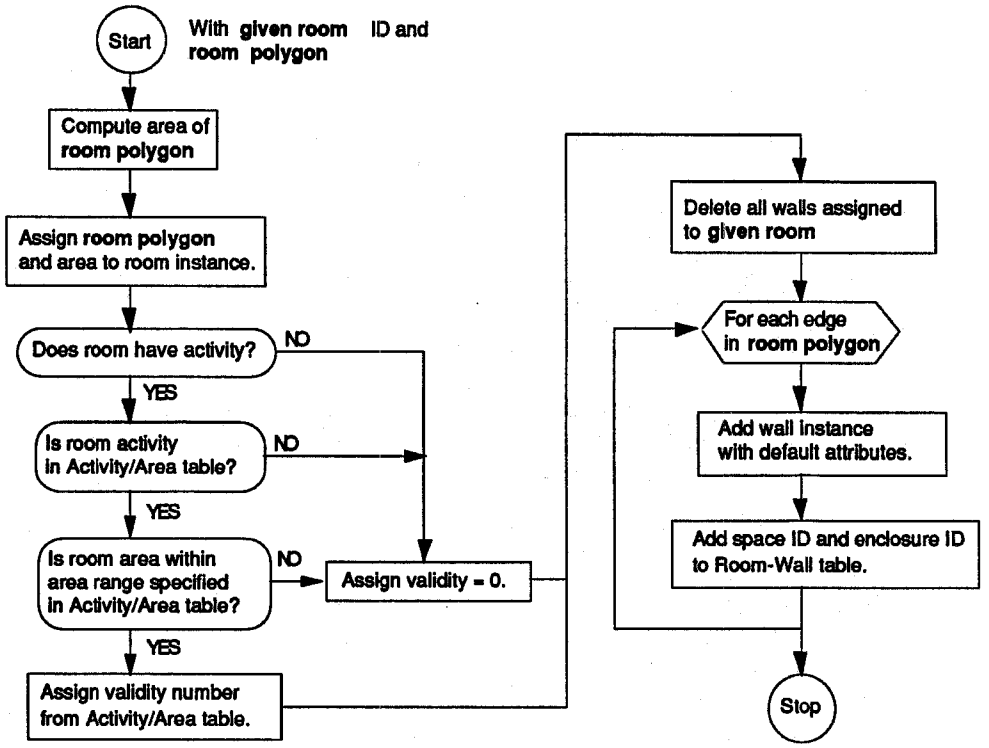


Figure 4.8.2 Data Base Activity 4

Entity Property Tables

The entity property tables can be indirectly derived from the conceptual model. For example:

a. Create a single table for each one-to-one, "has/of" bridge (in NIAM, a role between an object and a property with both a "total and "uniqueness" constraint):

OpeningIdentifier/DoorType, OpeningIdentifier/ProductID,
OpeningIdentifier/Location OpeningIdentifier/HingeLocation,
OpeningIdentifier/SwingDirection, OpeningIdentifier/HardwareType,
OpeningIdentifier/Finish, OpeningIdentifier/Size, OpeningIdentifier/JambType

b. "Join" the binary tables into a single "Door Property" table":

Opening Identifier	Door Type	Product ID	Location	Hinge Loc.	Swing Dir.	Hard. Type	Finish	Size	Jamb Type

Figure 5.1.1 Door Property Table

A new record (row) is added when a new building component is created, and default values are replaced with actual values as the building design proceeds through its stages of development.

In the design scenario example, there are four project dependent entity property tables needed to manage the building design information. The project independent data base (e.g., building assemblies data) will not be described in this section. Each entity table has a unique primary key for the RDBM system to access the data record, and zero, one, or more foreign keys might be included to establish relationships with other tables. The additional entity property tables are:

					Geometry		Env. Specs.		Design Req.	
Space Identifier	Name	Activity	Floor No.	Area	Room Poly.	...	Illum. Level	...	...	...

Figure 5.1.2 Room Property Table

					Geometry			Env. Specs.		Struct. Req.	
Enclosure Identifier	Wall Type	Mat. Assm.	Finish	Finish	Begin Point	End Point	Height	Acoustic Spec	...	...	...

Figure 5.1.3 Wall Property Table

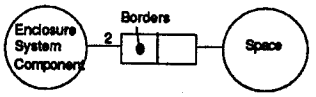
Furniture Identifier	Furn. Type	Loc.	Orient	ProdID

Figure 5.1.4 Furniture Property Table

Entity Relational Tables

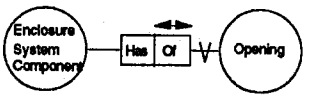
Entity relational tables are directly derived from the conceptual model, and arranged in a binary table. For example, a relational table can capture the "has-of" relationship between "Space" and "Enclosure System Component" in Figure 4.3.1 .

There are three relational tables are necessary to represent the many-to-many relationship of rooms, walls, doors and furniture:



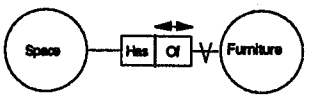
Space Identifier	Enclosure Identifier

Figure 5.2.1 – Room-Wall Relation Table
An Enclosure System Component borders exactly two Spaces



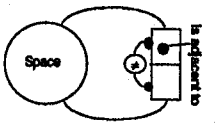
Enclosure Identifier	Opening Identifier

Figure 5.2.2 – Wall-Door Relation Table
An Opening is part of one and only one Enclosure System Component



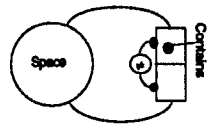
Space Identifier	Furniture Identifier

Figure 5.2.3 – Room-Furniture Relation Table
Furniture is part of one and only one Space



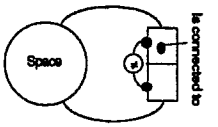
Space Identifier	Space Identifier

Figure 5.2.4 – Room Adjacency Relation Table
A Space may be adjacent to another space, but may not be adjacent to itself



Space Identifier	Space Identifier

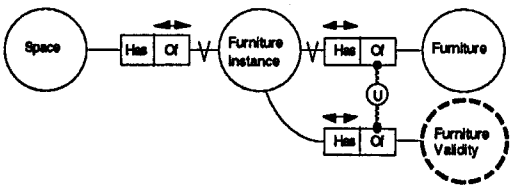
Figure 5.2.5 Room-Contains Relation Table
A Space may contain another space, but may not contain itself



Space Identifier	Space Identifier

Figure 5.2.6 Room-Connected Relation Table
A Space may be connected (by an opening) to another space, but may not be connected to itself

To support object validity values the following model and table can be used:



Space Identifier	Furniture Identifier	Furniture Validity

Figure 5.2.7 Room-Furniture-Validity Relation Table
A Furniture Instance is part of one and only one Space. Each Furniture Instance has one and only one Furniture (object) and one and only one Furniture Validity (number).

Space Identifier	Space Identifier	Validity

Figure 5.2.8 Room-Adjacency-Validity Relation Table

Object Oriented Data Base Design

In the OODBM approach, the information in the scenario is represented by two types of objects: Building System objects, which play a management role; and Building Component objects, which contain the internal states and behaviors of the rooms, walls, doors, and furniture. The states of an object class and the relationships between objects are based on the conceptual model, and the behaviors of the objects (to support the design actions) are implemented in the classes as methods.

Detail descriptions of the methods and instance variables are included fro the Space System, Space Objects and Room Object.

Building System Objects

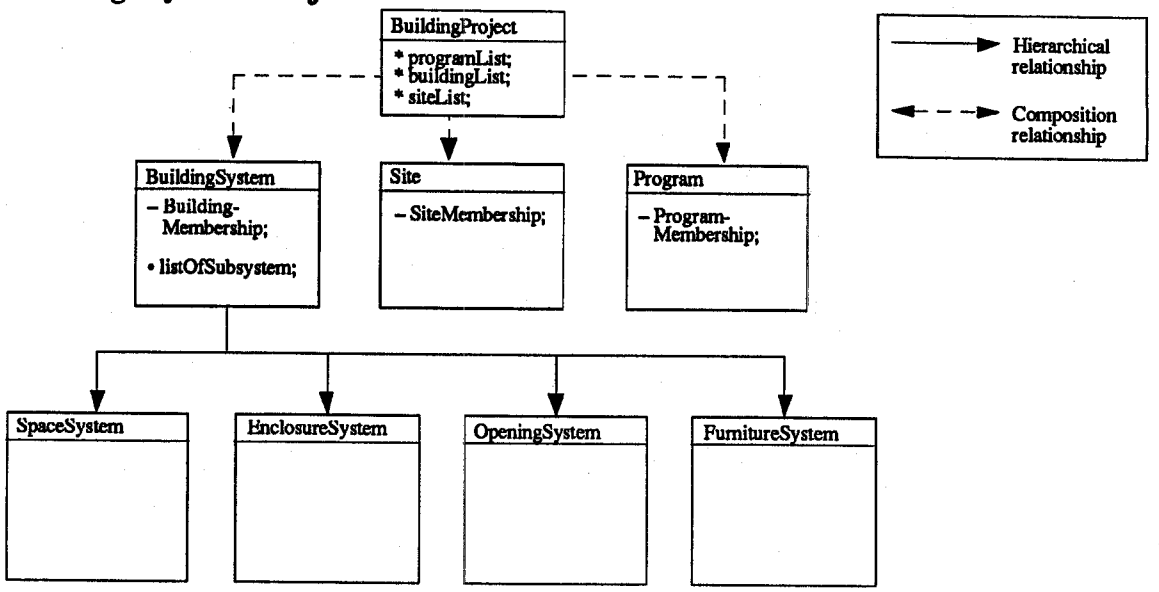


Figure 6.1.1 Building System Objects

SpaceSystem Type

Inherits from	BuildingSystem
Instance variables	
Methods	addFurniture:_intoRoom:_ ; addSpace:_intoSpace:_ ; changeNameOf:_to:_ ; checkCompatibilityBetweenFurniture:_ withDoor:_ ; checkConnectionBetween:_and:_ ; checkWallAdjacencyRequirement:_betweenRoom:_and:_ ; combineRoom:_into:_ ; create_new ; create_room ; put:_nextTo:_ ; removeFurnitureFrom:_ ; removeSharedWallBetween:_ and:_ ; resetShapeAndGeometryOf:_ ; setRoom:_validityValue:_ ;

Building Component Objects

The general states and methods which will be shared by subclasses are defined at the highest level class (e.g., Space), and more specific properties and methods are defined at the lower level classes (e.g. assigned, occupied, interior space).

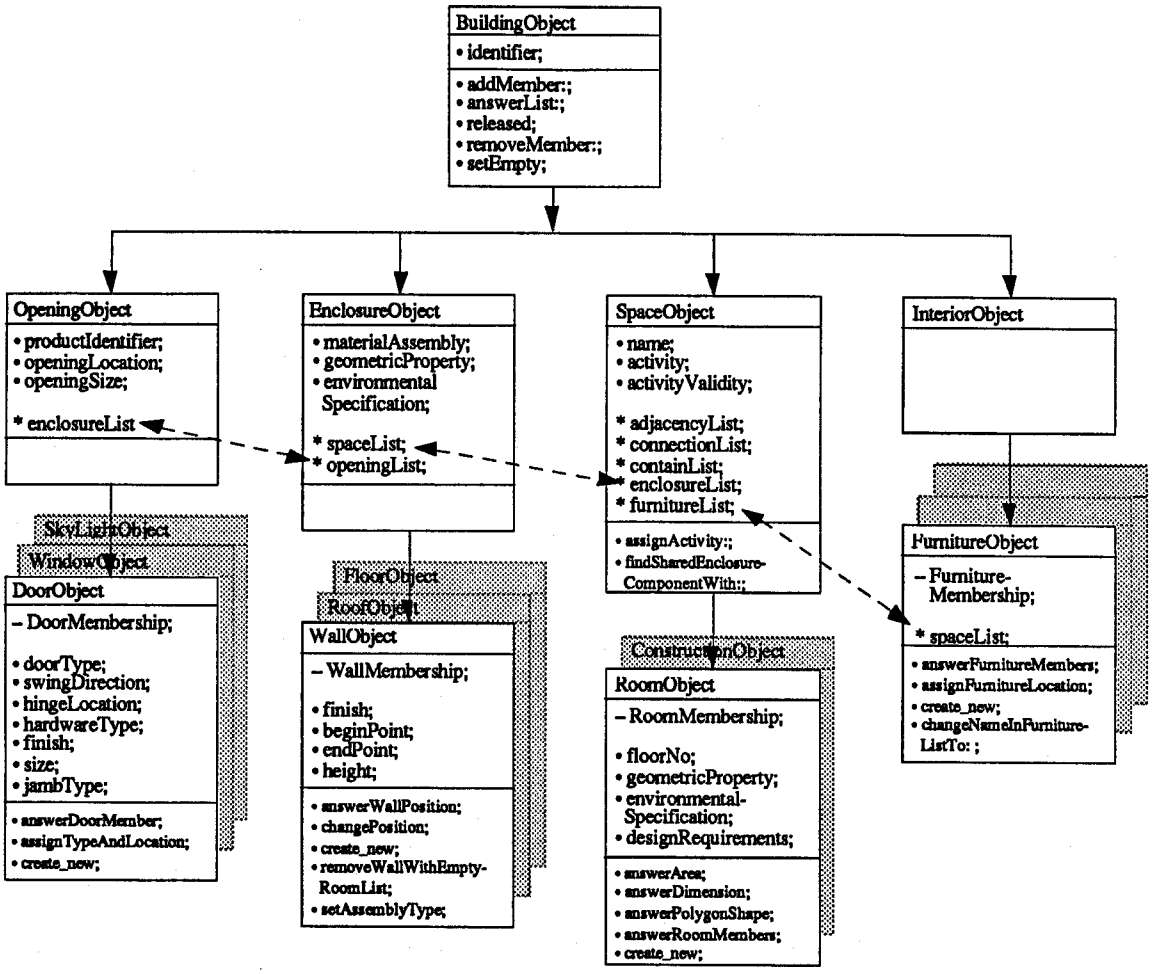


Figure 6.1.2 Building Component Objects

SpaceObject Type

This class defines the most general states and behaviors which are shared by other spatial subclasses. (Figure 4.1)

Inherits from	BuildingObject
Instance variables	name; activity; activityValidity;
	adjacencyList – a list structure for storing adjacent space;
	connectionList – a list structure for storing connected space;
	containList – a list structure for storing contained space;
	enclosureList – a list structure for storing walls;
	furnitureList – a list structure for storing furnitures.
Methods	assignActivity:_ ;
	findSharedEnclosureComponentWith:_ ;

RoomObject Type

The class defines room objects. (Figure 4.2)

Inherits from	Space

Class variable objects.	RoomMembership – a list structures for storing the membership of room
Instance variables	floorNo; geometricProperties – geometry attributes; environmentSpecifications – environmental specifications; designRequirements – design requirements;
Methods	answerArea ; answerDimension ; answerPloygonShape ; answerRoomMembers ; create_new ;

Relational Data Base Implementation

Because of the complex interrelationship of components in a building data base, it is not sufficient just to store and retrieve values. For example, in the scenario, the determination of room adjacency validity and wall assemblies will not be based on the properties of a single space or wall, but will be based on the relationship and activities of Rooms A and B, a variety of building program requirements, and possibly other types of general design guidelines. Any data base system for architectural design has to provide mechanisms to support these kinds of design considerations which are based on a network of dependencies.

Because data base tables defined in a traditional RDBM system are typically used in a passive manner, the analysis rules or procedures necessary to support building design activities need to be represented by external programming language or system-provided query and macro languages. As a result, any additions or modifications to the typical RDBM system command language needed to support rule and constraint checking must be in the form of customized operations imbedded in the data base system. Most commercial data base systems provide some programming ability such as, 4GL of INGRES™, PRO*C of ORACLE™, and the dBASE™ programming language.

There are two disadvantages of using a programming approach. First, the data base records and management (relational) operators must be interweaved with programming procedures which can cause the final system to be difficult to understand. Second, the system can be difficult to maintain by users without a programming background. And third, transferring building design analysis procedures into a programming language can be difficult.

Extended RDBM systems can ensure incoming and outgoing information is valid for the data base by applying domain candidacy procedures to the attributes of entity tables. (See [Cattell 91] for a discussion of extended RDBM systems.). Candidacy procedures [Borkin 88] can be any function which checks the validity of a value against a set of acceptance rules. The rules can be as simple as "any integer value", or "any integer number between a minimum and maximum value", or can be a function which compares the value using a complex algorithm. The candidacy procedures are sufficient for simple data checking (e.g., the maximum course number, valid birthday), or they can provide data checking and data computation based on complex domain relationships which are often necessary in architectural design.

The extended RDBM system used in this experiment was ArchModel [McIntosh, 84], a C and Unix-based system developed at the University of Michigan, College of Architecture.

It was used because of the ease of adding the necessary domain candidacy procedures (the latest version of the software was written in our lab by Turner, Borkin, et.al.).

The scenario was written in ArchModel's macro language. A portion of the macro file with brief explanations of the format is provided:

The format of the command for creating domains is "define domain-name data-type dimension domain-candidacy-procedure":

```
#Create domains.
  define SpaceID CHARACTER*8 1 none
  define RoomName CHARACTER*8 1 none
  define Activity CHARACTER*8 1 none
  define FloorNumber INTEGER*4 1 none
  define RoomPG2 GEOMETRY 1 DMAddRoomPG2
  define Area REAL*4 1 none
  define SpaceIDPair CHARACTER*8 2 none
  define Validity REAL*4 1 none
  define ActivityPair CHARACTER*8 2 none
  define AreaPair REAL*4 2 none
  ...
```

The format of the command for creating tables is "create table-name tuple-candidacy-procedure domain1 domain2 ...":

```
#   Create relations

  create RoomInstance none SpaceID RoomName Activity Area RoomPG2
  create RoomAdjacency none SpaceIDPair Validity
  ...
#   Create Building Program Relations
  create ActivityAdjacency none ActivityPair Validity
  create ActivityArea none Activity AreaPair
  ...
```

Values are added to the table using the ArchModel editor:

Design Action 0 Create building program data.base

```
edit ActivityAdjacency
  add-tuple office     office     1.
  add-tuple office     storage    1.
  add-tuple office     copy       .2
  add-tuple smalloffice smalloffice 1.
  add-tuple office     smalloffice 1.
stop
edit ActivityArea
  add-tuple smalloffice 80  120
  add-tuple office    700 1000
  add-tuple copy      300  500
stop
```

Design Action 1 Room A is added. Room B is added.

```
edit RoomInstance
  add-tuple 100 A none 1 none 0.
  add-tuple 101 B none 1 none 0.
stop
```

Design Action 2 Room A is adjacent to Room B.

```
edit RoomAdjacency
  add-tuple 100 101 1.
stop
```

Design Action 3 Room A given activity. Room B given activity.

```
edit RoomInstance
  find RoomName A
  add-value Activity office
  find RoomName B
  add-value Activity storage
stop
```

Design Action 4 Room A given polygonal shape. Room B given polygonal shape.

```
edit RoomInstance
  find RoomName A
  add-value RoomPG2 RoomA
  find RoomName B
  add-value RoomPG2 RoomB
stop
```

Object Oriented Data Base Implementation

Smalltalk-80 Release 4.0 on the Macintosh computer was used as the implementation environment for investigating the OODBM approach because of its easy prototyping ability. Smalltalk originated many of the definitions of today's object-oriented software development environments. We intend to implement a more complete example in Objective C using the environment of the NeXT computer.

Implementing the proposed conceptual model in object-oriented form is an iterative and incremental process. Several revisions were made during the testing because of the difficulties of finding proper classifications. Two sets of class hierarchies are established for implementing the system: the building system class hierarchy for managing building component objects; and the building component hierarchy for storing building component data. The main data types from the model are defined as class templates, and new objects are created by sending "create" message (e.g., create_room, create_wall) to their classes. The examples of the class definitions are listed in below.

The managerial classes – building system classes – provide essential functions to manage building components. For example, the enclosureSystem object maintains all instances of wallObject, roofObject, and floorObject. The managerial classes also contain methods for the purpose of candidacy checking. For example, spaceSystem contains the method used to check the adjacency of two rooms. Object relationships are defined as various lists. For example, roomObject contains a list of it associated enclosure objects.

System Initiation

```
spaceSystem := SpaceSystem new.
enclosureSystem := EnclosureSystem new.
buildingProgram := BuildingProgram new.
```

Design Action 1 Room A is added. Room B is added.

```
roomA := Room create_room.
roomB := Room create_room.
spaceSystem addSubsystem: roomA.
spaceSystem addSubsystem: roomB.
```

Two new Room object share created, and their identifiers are added to the subsystem list of spaceSystem for management purpose.

Design Action 2 Room A is adjacent to Room B.

```
spaceSystem put: roomA nextTo: roomB.
```

The spaceSystem method – put:nextTo: – is a procedures which updates the adjacencyList of room A and room B.

Design Action 3 Room A given activity. Room B given activity.

```
spaceSystem assignActivity: newActivity toRoom: roomA checkedBy: buildingProgram.
• room(a collection) := self findAdjacentRoomOf: roomA.
• adjacentActivities(a collection) := room(a collection) answerActivity.
• newValue(a list) := buildingProgram checkAdjacencyBetween: newActivity and: adjacentActivities(a
  collection).
• roomA setValidityValue: newValue.
spaceSystem assignActivity: newActivity toRoom: roomB checkedBy: buildingProgram.
```

The rooms adjacent to room A are found from the spaceSystem's listOfSubsystem. The activity of each adjacent room is retrieved. The room adjacency validity is checked based on the Activity/Adjacency pairs in the building program. A new validity value is assigned to room A.

Design Action 4 Room A given polygonal shape. Room B given polygonal shape.

```
roomA givenPolygonShape: roomPolygon.

lowHighArea(a pair) := buildingProgram answerRequiredLowHighAreaOf: roomA.
spaceSystem checkArea: roomA using: lowHighArea(a pair).

walls(for roomA) := Wall create_wall.
walls(for roomB) := Wall create_wall.
enclosureSystem addSubsystem: walls(for roomA).
enclosureSystem addSubsystem: walls(for roomB).
enclosureSystem addWall: walls(for roomA) toBorderSpace: roomA.
enclosureSystem addWall: walls(for roomB) toBorderSpace: roomB.
```

Comparison of RDBM and OODBM System Design

This paper poses two major aspects for the software designer to consider. First, the building systems as proposed in the NIAM model, need expression, and second, the design scenario actions need support. A traditional procedural designer, planning a C language implementation, might approach these aspects by translating the NIAM diagram into a collection of data type definitions, and the scenario into a collection of functions that implement the actions. The resulting program would support the actions, but would not provide a data base which could be shared by other processes. Also, it would be subject to

all of the maintenance and documentation difficulties that exist for custom-coded applications.

RDBM Observations

The relational approach that we used was the direct translation of the NIAM model into a number of relations, one for each major part of the diagram. The data integrity aspects of the scenario were expressed as domain candidacy procedures and the scenario actions as resulted in transactions on the relations in the data base. Implementing the proposed conceptual model in relational form was a simple task. The data from the model fit neatly into a set of tables and the design activities were easy to program with the macro language available in ArchModel.

Despite the ease of translation, the data base activities for the design actions were difficult to implement in a relational data base environment. There were two reasons for this. First, although ArchModel supports simple domain candidacy checking through the use of the C programming language and a collection of data base function calls (OpenRelation, OpenDomain, GetValue, PutValue, etc.), this capability was insufficient to support the complex dependencies inherent in building data; that is, adding or changing a single value in a table more often than not affects other values either in the same table or in different tables.

As a result, it was necessary to add the following domain candidacy capabilities to support the scenario:

a. Domain and single tuple only – The value can be compared to other values from the same tuple.
b. Domain and multiple tuples – A complex condition which might trigger the testing of attribute values from tuples from other relations.
c. Domain updating – A procedure which automatically updates a domain value from other values, either in the same tuple or in other tuples. An example is the computation of room area from a room polygon. Procedures of this type could be used to automatically update large portions of a data base.

Second, even with these data base candidacy extensions, providing the C functions to handle a data base activity was a complex programming task. This was mainly due to the tediousness and programming complexity of having to simultaneously deal with multiple relational tables through formal ArchModel function calls. It takes three or four function calls just to get a value from a table and three or four calls to put a value in a table. This is compared to a conventional C program where a single assignment statement is all that is necessary to get and put an array or structure value. The task was difficult even though the original ArchModel programmers (the authors of the paper) were available for the extensions. This approach would be even more difficult in most commercial RDBM systems since most of the necessary extensions are not supported.

OODBM Observations

The object oriented approach considers both the procedural aspects and the data aspects in a unified way. One design that we investigated and rejected paralleled the relational approach. In it we viewed a relation (a table) as a class or template for creating instances (the tuples of the relation). The class had methods (procedures for data integrity and transactions) that were inherited by the instances (the data for a table row). We found that we needed several additional class methods to manage the instances. On close examination of what was required we chose a design that separated the objects that managed from the

objects that represented the building system parts. This design allowed the separation of what was managed from how the managed objects responded to directions from the manager. The manager is viewed as an intelligent container of objects that respond to directions,. The building objects are viewed as objects that can implement management directions and other actions.

We observed two advantages in developing the OODBM approach. First, once a data base design is defined, the resulting conceptual diagram provides an easy and direct method for implementation and modification. These diagrams also become a valuable part of the documentation. They clearly show the instance variables, methods and inheritances that the implementation is based on.

Second, in contrast to the relational implementation, dependency networks were extremely easy to define and maintain, since changes of data went through object methods These methods, which set instance variables, in turn, send messages to dependent objects to upgrade their values. This, combined with two-way linking of objects, make data integrity and maintenance an inherent part of the objects.

Despite these clear advantages over the RDBM approach, object data base software is not widely available; therefore, a Smalltalk implementation is quite different than a C++ implementation and a commercial-based systems such as GemStone, ObjectStore and ORION.

Conclusion

Clearly, integrated data bases are needed for CAD system development. Can the built in operations and data structures of generic RDBM and OODBM systems support the dynamic activity of building design? Can they reflect the current status of a design as it moves through its stages from building programming to conceptual design to detailed design to construction planning and documentation? Can a data base maintain its correctness as a building design evolves? Does a data base system have built in mechanisms which allow it to respond to design changes? Building design is a dynamic process characterized by large amounts of interrelated data with degrees of exactness, constraints and goals which change over time.

Based on our work to date, we still believe that both the relational and object-oriented and the conventional procedural programming approaches offer much promise in the development of integrated CAD data bases. We plan to continue our efforts in these directions, and encourage others to share their experiences. It is only through joint efforts that this critical component of future CAD systems will come into being.

References

[Ahad 92] Ahad, R., Dedo, D., "OpenODB from Hewlett-Packard: a commercial object-oriented data base management system", *Journal of Object-Oriented Programming*, Vol. 4, No. 9, 1992.

[Bertino 91] Bertino, E. and Martino, L., *Object-Oriented Database Management Systems: Concepts and Issues*, 0018-9162/91/0400-0033, IEEE, 1991.

[Borkin 78] Borkin, H.J., McIntosh,J.F, McIntosh,P.G, Turner,J.A., "The development of three-dimensional spatial modeling techniques for the

construction planning of nuclear power plants", *Proceedings of SIGGRAPH 78*, Association for Computing Machinery, 1978

[Cattel 91] Cattel, R., *Object Data Management - Object-Oriented and Extended Relational Data base Systems*, Addison-Wesley, Massachusetts, 1991.

[Codd 79] Codd, E.F., "Extending the Database Relational Model to Capture More Meaning", in *ACM TODS*,4,4, Dec. 1979

[Cox 86] Cox, C.J., *Object-Oriented Programming - An Evolutionary Approach*, Addison-Wesley, Massachusetts, 1986.

[Date 86] Date, C.J., *An Introduction to Data base Systems*, Addison-Wesley, Massachusetts, 1986.

[Dahl 86] Dahl, O.J. and Nygaard, K., "SIMULA - an Algol-based Simulation Language", *Commun. ACM*, Vol. 9, Sept. 1966.

[Eastman 75] Eastman. C.M., *Spatial Synthesis in Computer-Aided Building Design*, John Wiley & Sons, 1975.

[Ketabchi 90] Ketabchi, M.A., Mathur, S., et al., *Comparative Analysis of RDBM systems and OODBM systems: A Case Study*, CH2843-1/90/0000/0528, IEEE, 1990.

[Kim 91] Kim, W., "Object-Oriented Data base Systems: Strength and Weakness", *The Journal of Object-Oriented Programming*, Vol. 4, No. 4, 1991.

[Kim 90] Kim, W., "Object-Oriented Data bases: Definition and Research Directions", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, No. 3, Sept. 1990.

[Kim 90] Kim, W., Garza, J.F., et al., "Architecture of the ORION Next-Generation Data base System", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, No. 1, March 1990.

[McIntosh, 84] McIntosh,J.F., *The Application of the Relational Data Model to Computer-Aided Building Design*, Doctoral Dissertation, The University of Michigan, 1984.

[Shlaer 88] Shlaer, S., Mellor, S.J., *Object-Oriented Systems Analysis - Modeling the World in Data*, Yourdon Press, New Jersey, 1988.

[Turner 90] Turner,J.A, *AEC Building Systems Model*, ISO TC184/SC4/WG1 (STEP) working paper August 1990.

[Turner 90] Turner,J.A, *Guide to Reading NIAM Diagrams*, The University of Michigan, October 1990.

[van Griethuysen 92] van Griethuysen, J.J., *Concepts and Terminology for the Conceptual Schema and the Information Base*, ISO/TC97/SC5 - N695